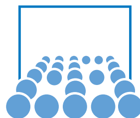
HPC – Algorithms and Applications

Dwarf # 1 – Dense Linear Algebra

Michael Bader

Winter 2013/2014



The Seven Dwarfs of HPC – Dwarf # 1

1. **dense linear algebra**
2. sparse linear algebra
3. spectral methods
4. N-body methods
5. structured grids
6. unstructured grids
7. Monte Carlo

Part I

Matrix Multiplication

Matrix Multiplication on the PRAM

```
MatrixMult_PRAM(A:Matrix[n], B:Matrix[n], C:Matrix[n]) {
  ! Model: CREW−PRAM with n*n processors
  for i from 1 to n do in parallel
    for k from 1 to n do in parallel
      for j from 1 to n do
        C[i,k] = C[i,k] + A[i,j]*B[j,k]
}
```

Strategy:

- each processor computes one element of the result matrix
  $\Rightarrow$ exclusive write access to C[i,k]
- runtime $O(n)$ on n^2 processors

Matrix Multiplication on the PRAM

```
MatrixMult_PRAM(A:Matrix[n], B:Matrix[n], C:Matrix[n]) {
  ! Model: CREW-PRAM with n*n processors
  for i from 1 to n do in parallel
    for k from 1 to n do in parallel
      for j from 1 to n do
        C[i,k] = C[i,k] + A[i,j]*B[j,k]
}
```

Observation:

- synchronised execution of the for-j-loop;
  $\rightarrow$ in the first step:
- all processors $P_{i,k}$ simultaneously access A[i,1]
- all processors $P_{i,k}$ simultaneously access B[1,k]

PRAM Matrix Multiplication with Shifted Indices

```
MatrixMult_PRAM(A:Matrix[n], B:Matrix[n], C:Matrix[n]) {
  ! Model: EREW−PRAM with n∗n processors
  for i from 1 to n do in parallel
    for k from 1 to n do in parallel
      for j from 1 to n do
        C[i,k] = C[i,k]
                 + A[i,(i+j+k) mod n]∗B[(i+j+k) mod n,k]
}
```

Observation: in the first step of the j-loop

- processors $P_{i,k}$ access A[i,(i+1+k) mod n]
- processors $P_{i,k}$ access B[(i+1+k) mod n,k]
- leads to exclusive read access to A and B

Towards Cannon's Algorithm

Strategy:

- consider a 2D mesh of processors
- each processor holds one element of A, B, and C
- C element stays on a fixed processor
- during j-loop:
  increase column index of A, and row index of B
  $\Rightarrow$ A and B element have to be "transferred" to neighbouring processors

New Algorithmic Model:

Interconnection Network (2D Mesh) instead of PRAM

Matrix Multiplication on the 2D Mesh

→ **"Cannon's Algorithm":**

```
MatrMult_2Dmesh(A:Matrix[n], B:Matrix[n], C:Matrix[n]) {
  ! Model: 2D mesh with n*n processors
  input P[i,k]:A,B,C, range P[i,k]:  1<=i,k<=n
  !  init :  P[i,k]  holds A[i,( i+k+1)mod n], B[(i+1+k) mod n,k]
  for  P[i,k]:  1<=i,k<=n do in parallel
    for  j  from 1 to n do {
      C = C + A*B;
      P[i,k]:A <<< P[i,(k+1) mod n]:A
      P[i,k]:B <<< P[(i+1) mod n,k]:B
    }
   end in parallel
   output P[i,k]:C, range P[i,k]:  1<=i,k<=n
}
```

Cannon's Algorithm – Time Complexity

Complexity of Cannon's Algorithm:

- n additions/multiplications on n^2 processors
- $2n$ send/receive operations (by each processor)

Cannon's Algorithm – Time Complexity

Complexity of Cannon's Algorithm:

- n additions/multiplications on n^2 processors
- $2n$ send/receive operations (by each processor)

Generalization:

- switch to block-oriented algorithm on p^2 processors
- A[i,j], B[j,k], C[i,k] are matrix blocks (of size $\frac{n}{p} \times \frac{n}{p}$)
- leads to $p(n/p)^3 = n^3/p^2$ operations
- $2p$ block send/receives – total volume: $2n^2/p$ elements

Cannon's Algorithm – Time Complexity

Complexity of Cannon's Algorithm:

- n additions/multiplications on n^2 processors
- $2n$ send/receive operations (by each processor)

Generalization:

- switch to block-oriented algorithm on p^2 processors
- A[i,j], B[j,k], C[i,k] are matrix blocks (of size $\frac{n}{p} \times \frac{n}{p}$)
- leads to $p(n/p)^3 = n^3/p^2$ operations
- $2p$ block send/receives – total volume: $2n^2/p$ elements

Mesh $\rightarrow$ Torus:

- mistake in the analysis: transfer P[i,n]:A $<<<$ P[i,1]:A (similar for B) not possible in one step
- actually requires a **2D torus** network

Matrix Multiplication and Parallel External Memory

Consider: "Parallel External Memory" Model

- number of memory transfers for the PRAM program?

```
for i from 1 to n do in parallel
  for k from 1 to n do in parallel
    for j from 1 to n do
      C[i,k] = C[i,k] + A[i,j]*B[j,k]
```

Matrix Multiplication and Parallel External Memory

Consider: "Parallel External Memory" Model

- number of memory transfers for the PRAM program?

```
for i from 1 to n do in parallel
  for k from 1 to n do in parallel
    for j from 1 to n do
      C[i,k] = C[i,k] + A[i,j]*B[j,k]
```

- each processor reads $2n$ elements
- thus $n^2 \cdot 2n/L$ transfers (of cache lines, each with L words)
- how to use the local memory?

Block-Oriented Matrix Multiplication

Block-oriented formulation:

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{pmatrix}$$

$\rightarrow$ each A_{ij}, B_{jk}, C_{ik} a square matrix block

Block operations:

- $C_{11} = A_{11}B_{11} + A_{12}B_{21} + A_{13}B_{31}$
  $C_{12} = A_{11}B_{12} + A_{12}B_{22} + A_{13}B_{32}$
  $\ldots$
- three blocks need to fit in local memory!

Matrix Multiplication on Parallel External Memory

```
MatrixMult_PRAM(A:Matrix[n], B:Matrix[n], C:Matrix[n]) {
  ! Model: CREW−PRAM with (n/b)*(n/b) processors
  for i from 0 to n/b−1 do in parallel
    for k from 0 to n/b−1 do in parallel
      for j from 0 to n/b−1 do {
        ! read required blocks of A and B (also C for j=1)
        for ii from 1 to b do
          for kk from 1 to b do
            for jj from 1 to b do
              C[i*b+ii,k*b+kk] = C[i*b+ii,k*b+kk]
                                 + A[i*b+ii,j*b+jj]*B[j*b+jj,k*b+kk]
      }
}
```

Blocked MatrixMult on PEM – Memory Transfers

Number of Memory Transfers:

- local memory can hold M words;
  choose b such that $3b^2 < M$ (ideal: $3b^2 = M$)
- read/write local C block: $2b^2$ words per processor
- read all n/b A blocks and n/b B blocks:
  $2n/b \cdot b^2$ words per processor
- for all $(n/b)^2$ processors:
  $(n/b)^2 \cdot (2n/b + 2)b^2 = 2n^3/b + 2n^2$ words
- $b \in \Theta(\sqrt{M})$; move L words per transfer;
  therefore: $\Theta(n^3/(L\sqrt{M}))$ memory transfers

Block-Oriented and Block-Recursive Multiplication

Block-oriented algorithms:

$$C_{ik} = \sum_j A_{ij} B_{jk}$$

Block-recursive algorithms:

$$\left(\begin{array}{cc|cc} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ \hline A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{array}\right) \left(\begin{array}{cc|cc} B_{11} & B_{12} & B_{13} & B_{14} \\ B_{21} & B_{22} & B_{23} & B_{24} \\ \hline B_{31} & B_{32} & B_{33} & B_{34} \\ B_{41} & B_{42} & B_{43} & B_{44} \end{array}\right) = \left(\begin{array}{cc|cc} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ \hline C_{31} & C_{32} & C_{33} & C_{34} \\ C_{41} & C_{42} & C_{43} & C_{44} \end{array}\right)$$

$\rightarrow$ blocking leads to better cache performance

Part II

Dense Linear Algebra

Dense Linear Algebra – Libraries

BLAS

- "Basic linear algebra subroutines for FORTRAN usage" (Lawson et al., ACM TOMS, 1979)
- BLAS level 3: dense matrix multiplication, focus on blocking to improve cache efficiency (Dongarra et al., 1990)

LAPACK

- "a portable linear algebra library for high-performance computers" (1990)
- solve systems of linear equations, least squares problems, eigenvalue problems, etc.

Dense Linear Algebra – Parallelisation

PBLAS

- parallel BLAS implementation (OpenMP, MPI)
- basis for ScaLAPACK

ScaLAPACK

- parallel library (subset of LAPACK) for distributed memory computers (v1.0: 1995; v1.8: 2007)
- basic data structure: 2D block cyclic decomposition of matrices

Example: LU Decomposition

Block-oriented formulation:

$$\begin{pmatrix} \widehat{L}_{11} & 0 & 0 \\ L_{21} & \widehat{L}_{22} & 0 \\ L_{31} & L_{32} & \widehat{L}_{33} \end{pmatrix} \begin{pmatrix} \widetilde{U}_{11} & U_{12} & U_{13} \\ 0 & \widetilde{U}_{22} & U_{23} \\ 0 & 0 & \widetilde{U}_{33} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix},$$

Block operations:

1. LU decomposition: $\widehat{L}_{11}\widetilde{U}_{11} = A_{11}$
2. “find left”: $L_{21}\widetilde{U}_{11} = A_{21}$, $L_{31}\widetilde{U}_{11} = A_{31}$
3. “find right”: $\widehat{L}_{11}U_{12} = A_{12}$, $\widehat{L}_{11}U_{13} = A_{13}$
4. block updates: $A_{22} = A_{22} - L_{21}U_{12}$

etc.

Barrier/Tile-Oriented Parallelisation

$$L_{11}U_{11} = A_{11}$$

$L_{21}U_{11} = A_{21}$ | $L_{31}U_{11} = A_{31}$ | $L_{11}U_{12} = A_{12}$ | $L_{11}U_{13} = A_{13}$

$A_{23} -= L_{21}U_{13}$ | $A_{22} -= L_{21}U_{12}$ | $A_{32} -= L_{31}U_{12}$ | $A_{33} -= L_{31}U_{13}$

$$L_{22}U_{22} = A_{22}$$

$L_{32}U_{22} = A_{32}$ | $L_{22}U_{23} = A_{23}$

$$A_{33} -= L_{32}U_{23}$$

$$L_{33}U_{33} = A_{33}$$

Panel Updates

Standard situation for panel update (L and U stored in-place!):

$$\left(\begin{array}{c|c|c} \widehat{L}_{11} & 0 & 0 \\ \hline L_{21} & I & 0 \\ \hline L_{31} & 0 & I \end{array}\right) \left(\begin{array}{c|c|c} \widetilde{U}_{11} & U_{12} & U_{13} \\ \hline 0 & A^*_{22} & A^*_{23} \\ \hline 0 & A^*_{32} & A^*_{33} \end{array}\right)$$

- $\widehat{L}_{11}$, $\widetilde{U}_{11}$: factorized block, stored in-place
- U_{12}, U_{13}, L_{21}, L_{31} updates already computed
- next step: LU-decomposition on current tile A^*_{22}
- then: panel updates to compute U_{23}, L_{31}
- update trailing matrix A^*_{33} and move to next tile

Tile-Oriented Parallelisation

Tile-oriented LU decomposition

- successive panel updates on tiles
- use BLAS-2 and BLAS-3 operations for updates (leads to cache blocking)

For parallel LU decomposition (assume distributed memory):

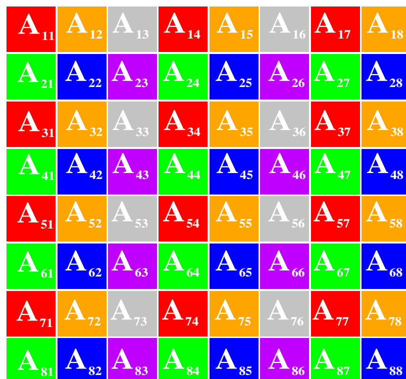
- how to place tiles on parallel processes
- ... to ensure good load balance
- ... to retain BLAS-2 and BLAS-3 operations

Candidates for tile distribution:

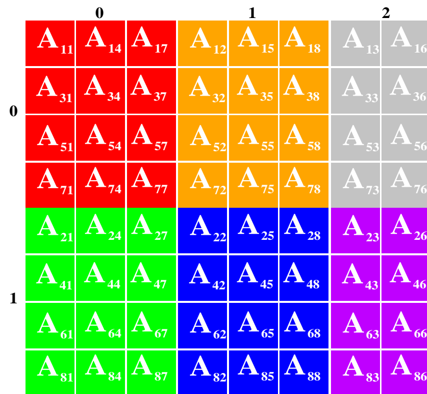
- row/column-wise?
- row/column-cyclic?
- block-wise or block-cyclic?

2D Block Cyclic Decomposition

matrix view:

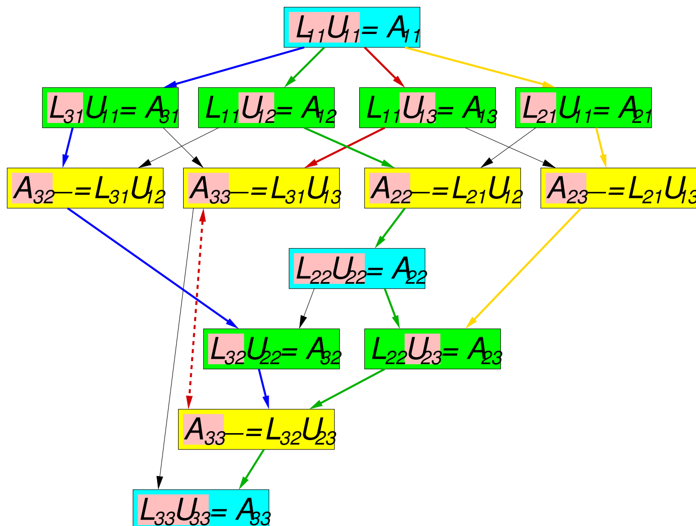


process view:



(source: ScaLAPACK)

Graph-Oriented Parallelisation



Current Library Projects

PLASMA (`http://www.netlib.org/plasma/`):

- new library project to replace ScaLAPACK
- block-oriented algorithms for dense linear algebra
- DAG-based scheduling of operations
- MAGMA: version for GPUs

FLAME (`http://z.cs.utexas.edu/wiki/flame.wiki/`):

- “formal linear algebra methods environment”
- block-oriented algorithms for dense linear algebra
- based on block-oriented notation of algorithms; concept to transfer notation into program code

Current Library Projects (2)

SMP superscalar
(`http://www.bsc.es/plantillaG.php?cat_id=385`):

- pragma-based programming model (similar to OpenMP)
- input/output qualifiers to declare data dependencies
- scheduling of operations according to DAG

Part III

Matrix Multiplication with Peano Space-Filling Curves

Multiplication of 3×3 Matrices

An optimal order of execution:

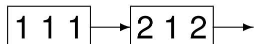
$$\begin{pmatrix} \mathbf{1} & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} \begin{pmatrix} \mathbf{1} & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} = \begin{pmatrix} \mathbf{1} & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix}$$

1 1 1 →

Multiplication of 3×3 Matrices

An optimal order of execution:

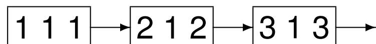
$$\begin{pmatrix} 1 & 6 & 7 \\ \mathbf{2} & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} \begin{pmatrix} \mathbf{1} & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 6 & 7 \\ \mathbf{2} & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

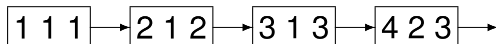
$$\begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ \mathbf{3} & 4 & 9 \end{pmatrix} \begin{pmatrix} \mathbf{1} & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ \mathbf{3} & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

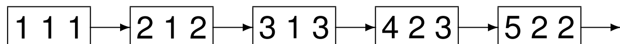
$$\begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ 3 & \mathbf{4} & 9 \end{pmatrix} \begin{pmatrix} 1 & 6 & 7 \\ \mathbf{2} & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ \mathbf{3} & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

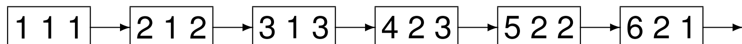
$$\begin{pmatrix} 1 & 6 & 7 \\ 2 & \mathbf{5} & 8 \\ 3 & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 & 6 & 7 \\ \mathbf{2} & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 6 & 7 \\ \mathbf{2} & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

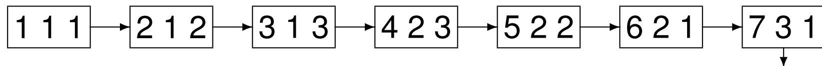
$$\begin{pmatrix} 1 & \mathbf{6} & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 & 6 & 7 \\ \mathbf{2} & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} = \begin{pmatrix} \mathbf{1} & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

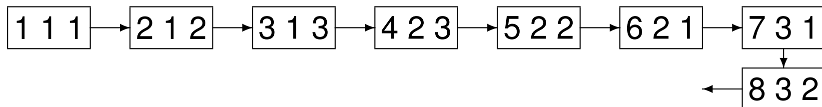
$$\begin{pmatrix} 1 & 6 & \mathbf{7} \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ \mathbf{3} & 4 & 9 \end{pmatrix} = \begin{pmatrix} \mathbf{1} & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

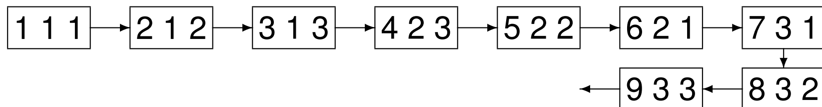
$$\begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & \mathbf{8} \\ 3 & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ \mathbf{3} & 4 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 6 & 7 \\ \mathbf{2} & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

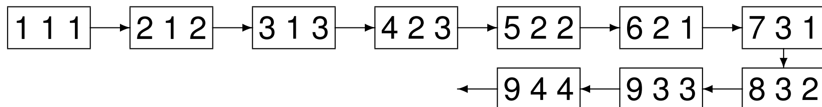
$$\begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & \mathbf{9} \end{pmatrix} \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ \mathbf{3} & 4 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ \mathbf{3} & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

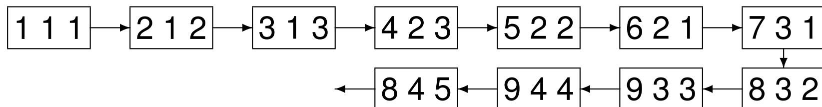
$$\begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & \mathbf{9} \end{pmatrix} \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ 3 & \mathbf{4} & 9 \end{pmatrix} = \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ 3 & \mathbf{4} & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

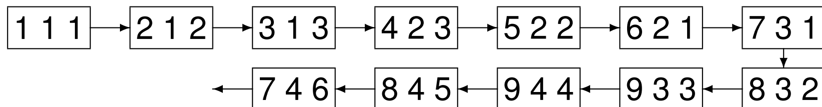
$$\begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & \mathbf{8} \\ 3 & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ 3 & \mathbf{4} & 9 \end{pmatrix} = \begin{pmatrix} 1 & 6 & 7 \\ 2 & \mathbf{5} & 8 \\ 3 & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

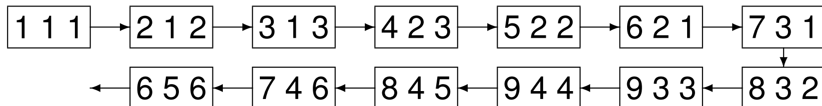
$$\begin{pmatrix} 1 & 6 & \mathbf{7} \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ 3 & \mathbf{4} & 9 \end{pmatrix} = \begin{pmatrix} 1 & \mathbf{6} & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

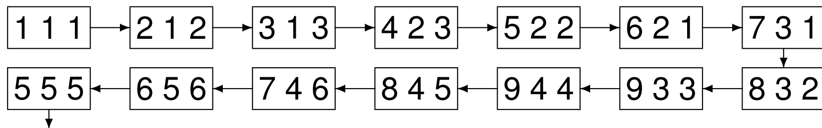
$$\begin{pmatrix} 1 & \mathbf{6} & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 & 6 & 7 \\ 2 & \mathbf{5} & 8 \\ 3 & 4 & 9 \end{pmatrix} = \begin{pmatrix} 1 & \mathbf{6} & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

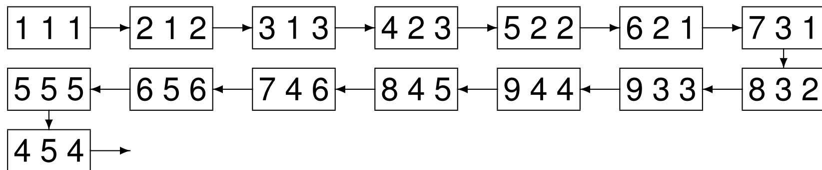
An optimal order of execution:

$$\begin{pmatrix} 1 & 6 & 7 \\ 2 & \mathbf{5} & 8 \\ 3 & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 & 6 & 7 \\ 2 & \mathbf{5} & 8 \\ 3 & 4 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 6 & 7 \\ 2 & \mathbf{5} & 8 \\ 3 & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

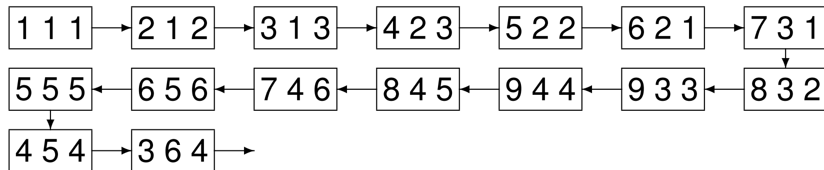
An optimal order of execution:



Multiplication of 3×3 Matrices

An optimal order of execution:

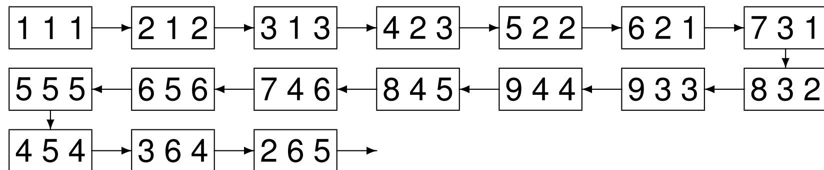
$$\begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ \mathbf{3} & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 & \mathbf{6} & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ 3 & \mathbf{4} & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

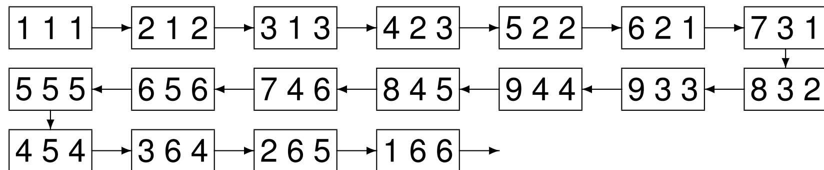
$$\begin{pmatrix} 1 & 6 & 7 \\ \mathbf{2} & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 & \mathbf{6} & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 6 & 7 \\ 2 & \mathbf{5} & 8 \\ 3 & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

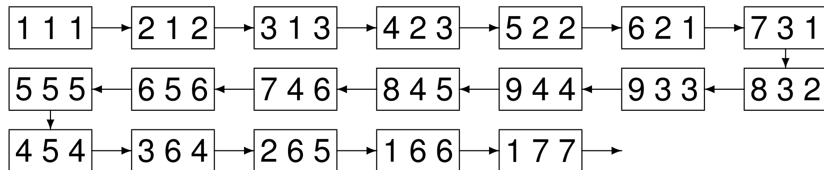
$$\begin{pmatrix} \mathbf{1} & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 & \mathbf{6} & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} = \begin{pmatrix} 1 & \mathbf{6} & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

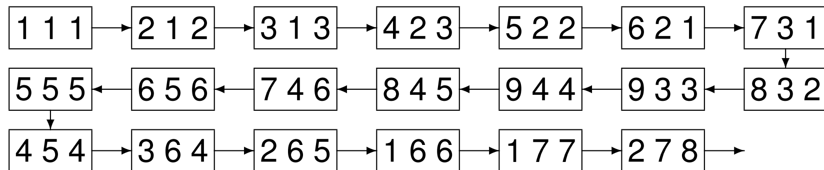
$$\begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

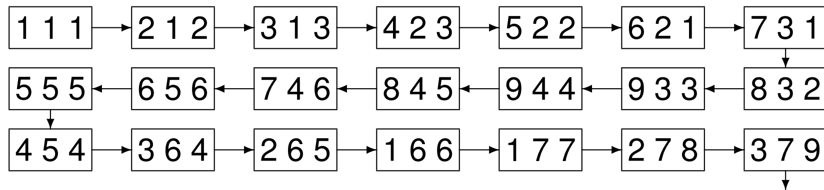
$$\begin{pmatrix} 1 & 6 & 7 \\ \mathbf{2} & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 & 6 & \mathbf{7} \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & \mathbf{8} \\ 3 & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

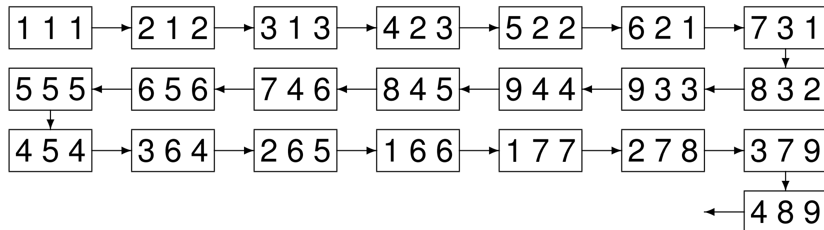
$$\begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ \mathbf{3} & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 & 6 & \mathbf{7} \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & \mathbf{9} \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

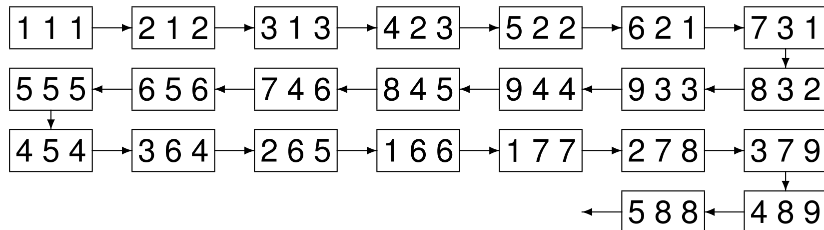
$$\begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ 3 & \mathbf{4} & 9 \end{pmatrix} \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & \mathbf{8} \\ 3 & 4 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & \mathbf{9} \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

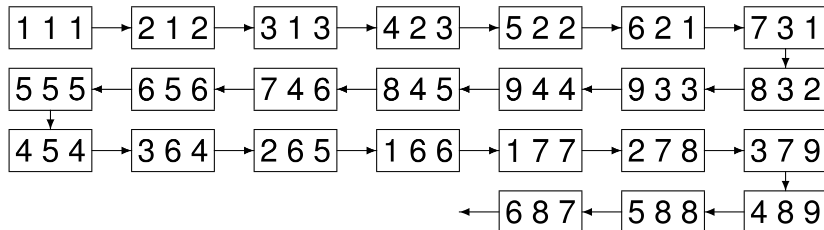
$$\begin{pmatrix} 1 & 6 & 7 \\ 2 & \mathbf{5} & 8 \\ 3 & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & \mathbf{8} \\ 3 & 4 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & \mathbf{8} \\ 3 & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

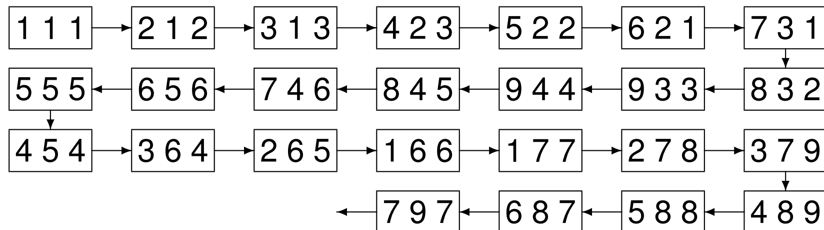
$$\begin{pmatrix} 1 & \mathbf{6} & 7 \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & \mathbf{8} \\ 3 & 4 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 6 & \mathbf{7} \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

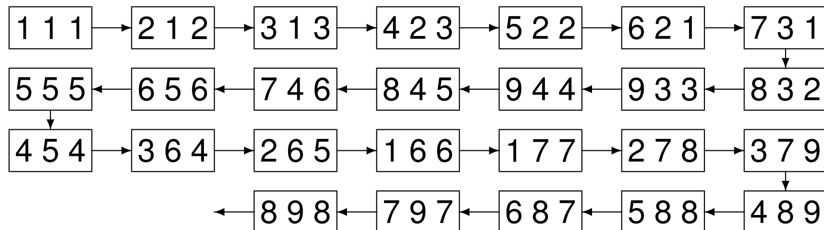
$$\begin{pmatrix} 1 & 6 & \mathbf{7} \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & \mathbf{9} \end{pmatrix} = \begin{pmatrix} 1 & 6 & \mathbf{7} \\ 2 & 5 & 8 \\ 3 & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

An optimal order of execution:

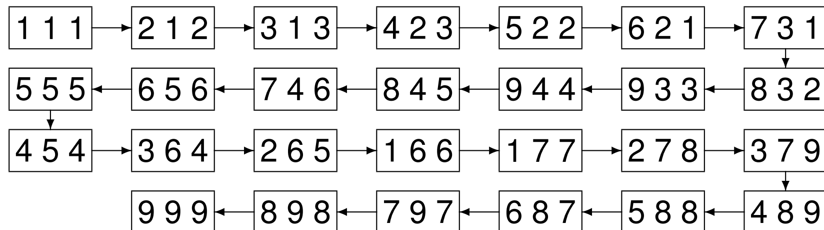
$$\begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & \colorbox{green}{$\mathbf{8}$} \\ 3 & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & \colorbox{green}{$\mathbf{9}$} \end{pmatrix} = \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & \colorbox{green}{$\mathbf{8}$} \\ 3 & 4 & 9 \end{pmatrix}$$



Multiplication of 3×3 Matrices

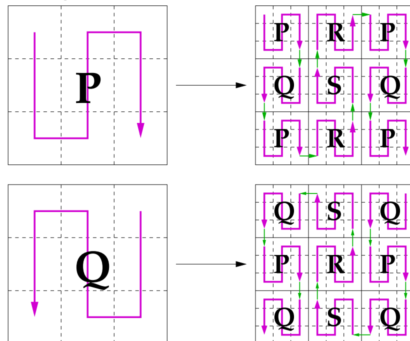
An optimal order of execution:

$$\begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & \mathbf{9} \end{pmatrix} \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & \mathbf{9} \end{pmatrix} = \begin{pmatrix} 1 & 6 & 7 \\ 2 & 5 & 8 \\ 3 & 4 & \mathbf{9} \end{pmatrix}$$



Block-recursive Peano Element Order

- indexing according to iteration of a Peano curve



- in practice: stopped on *L1 blocks*
  (size tuned to L1 cache $\rightarrow$ 2 matrix blocks should fit)
- use column-major layout within L1-blocks

Block-recursive Multiplication

- multiplication of block matrices (cmp. 3×3-scheme):

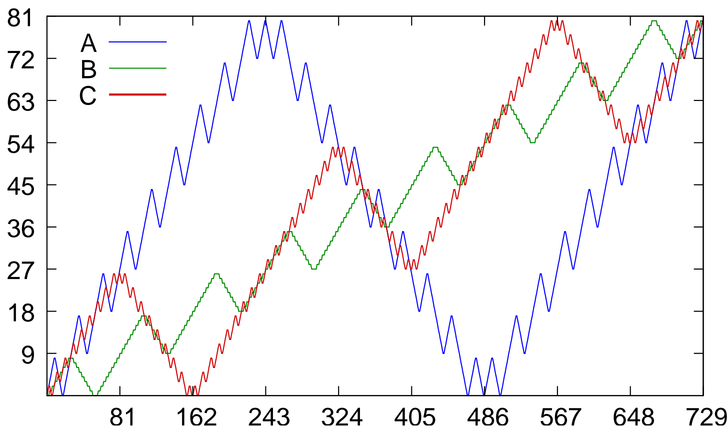
$$\begin{pmatrix} P_{A0} & R_{A5} & P_{A6} \\ Q_{A1} & S_{A4} & Q_{A7} \\ P_{A2} & R_{A3} & P_{A8} \end{pmatrix} \begin{pmatrix} P_{B0} & R_{B5} & P_{B6} \\ Q_{B1} & S_{B4} & Q_{B7} \\ P_{B2} & R_{B3} & P_{B8} \end{pmatrix} = \begin{pmatrix} P_{C0} & R_{C5} & P_{C6} \\ Q_{C1} & S_{C4} & Q_{C7} \\ P_{C2} & R_{C3} & P_{C8} \end{pmatrix}$$

- leads to **eight** different combinations (w.r.t. block numbering):

$$\begin{array}{cccc} PP \rightarrow P & QR \rightarrow S & RS \rightarrow R & SQ \rightarrow Q \\ PR \rightarrow R & QP \rightarrow Q & RQ \rightarrow P & SS \rightarrow S \end{array}$$

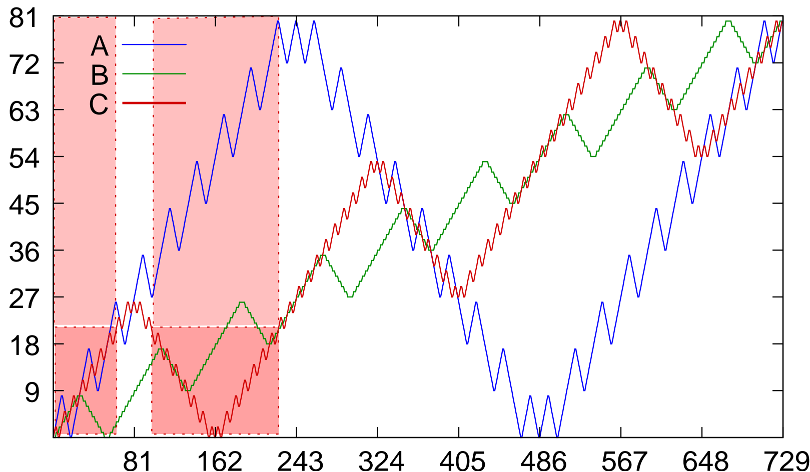
- all schemes similar to $PP \rightarrow P$ (but revers order)

Access Pattern to the Matrix Blocks



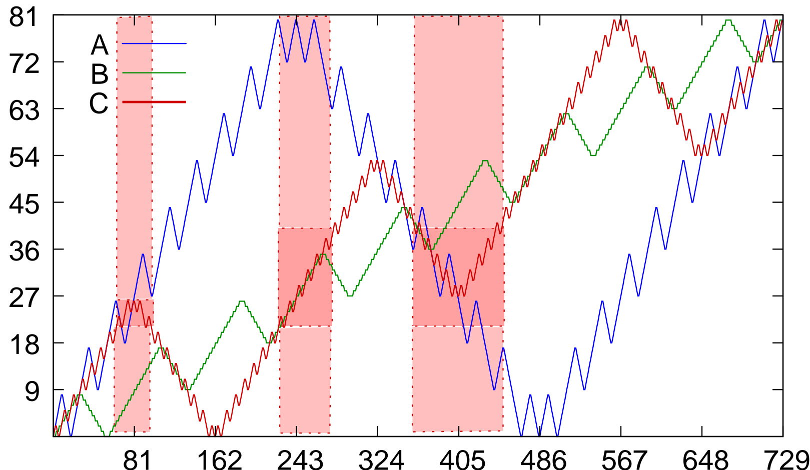
- Increment/Decrement access to elements
- $\mathcal{O}(k^3)$ operations on any block of k^2 elements

Parallelisation: *Owner-Computes*-Partitioning



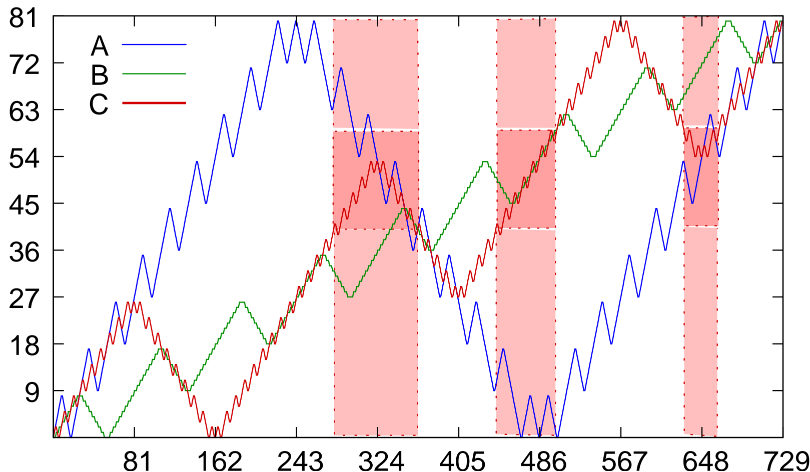
Partition matrix C according to Peano order → defines thread

Parallelisation: *Owner-Computes*-Partitioning



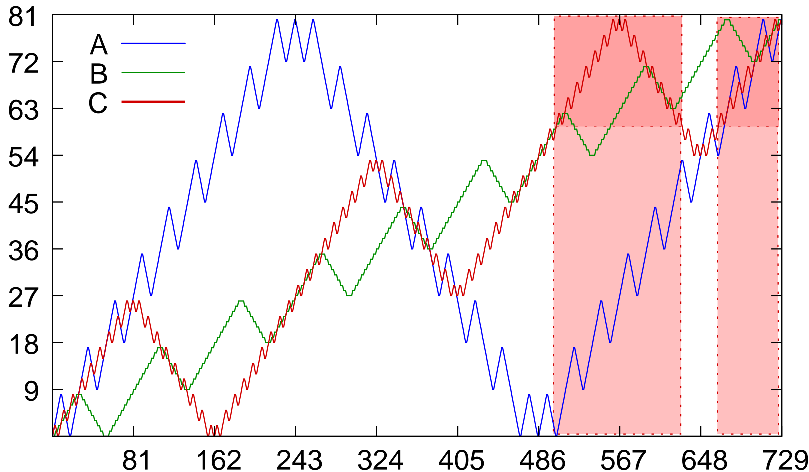
Partition matrix C according to Peano order $\rightarrow$ defines thread

Parallelisation: *Owner-Computes*-Partitioning



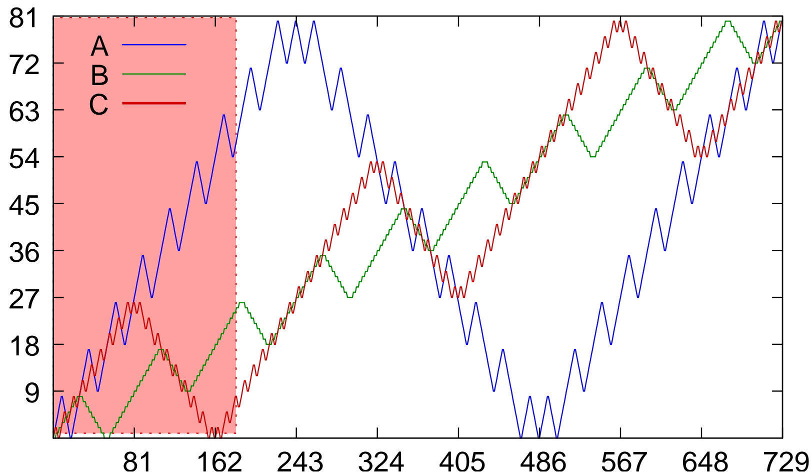
Partition matrix C according to Peano order → defines thread

Parallelisation: *Owner-Computes*-Partitioning



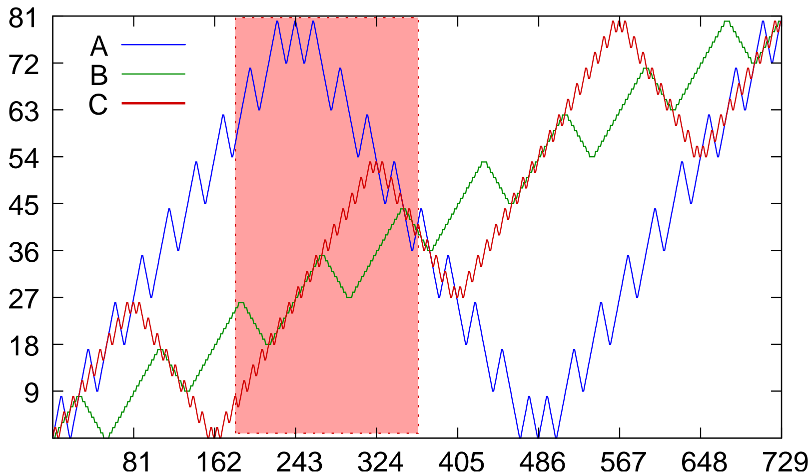
Partition matrix C according to Peano order → defines thread

Parallelisation: *Work-Oriented*-Partitioning



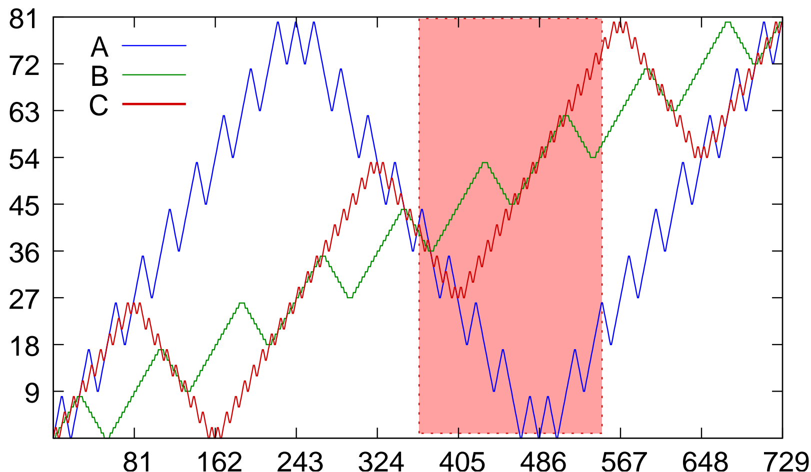
Partition element operations according to 3D Peano order
$\rightarrow$ defines thread

Parallelisation: *Work-Oriented*-Partitioning



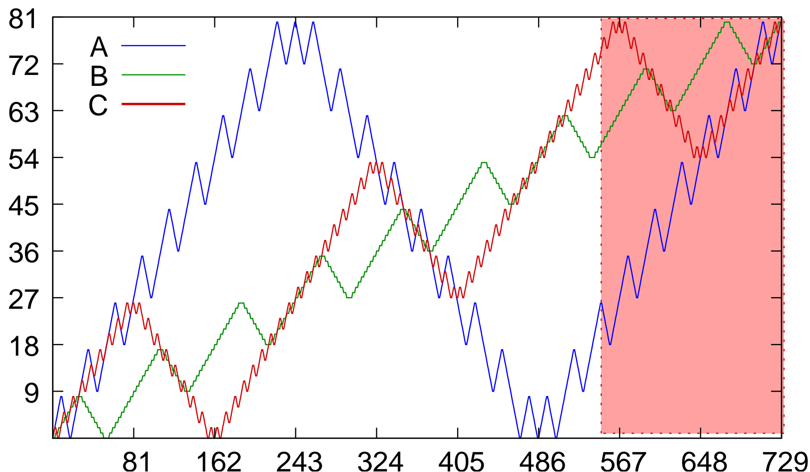
Partition element operations according to 3D Peano order
$\rightarrow$ defines thread

Parallelisation: *Work-Oriented*-Partitioning



Partition element operations according to 3D Peano order
$\rightarrow$ defines thread

Parallelisation: *Work-Oriented*-Partitioning



Partition element operations according to 3D Peano order
$\rightarrow$ defines thread

"Peano" Multiplication – Properties

- access to matrix elements entirely local:
  use increment and decrement operators only
- optimal re-use of data:
  $\mathcal{O}(k^3)$ **operations on any block of** k^2 **contiguous elements**
- number of cache misses on an *ideal cache* (M lines of length L):

$$\mathcal{O}\left(\frac{N^3}{L\sqrt{M}}\right)$$

  shown to be asymptotically optimal
- results can be extended to Parallel External Memory model